Paper 4008

Middleware - The Missing Link in Client-Server Migration

William Carter
Southeastern Data Cooperative, Inc.
2100 East Exchange Place
Tucker, Georgia 30084-5313
c/o Russ Rino
LeeTech Software
(408) 253-1987

Background

Southeastern Data Cooperative was established in 1976 as a turnkey supplier of data processing hardware and software solutions to the Rural Electric Membership Cooperative market. We presently have over 100 HP 3000 installations at member corporations located across the United States from Alaska to Florida. These stand-alone systems range in size from the 918 through the 987 and all are running the full suite of electric utility billing and accounting applications developed, maintained and supported by SEDC on behalf of its member companies. In these constantly evolving mission-critical applications, three large and complex TurboImage databases have been developed to support several hundred batch and interactive programs using COBOL and View running on 'dumb' terminals.

The User Mandate

It goes without saying that over such a long period of time, in such a highly regulated industry spanning 49 states and serving over 100 independent companies, the data structures and supporting code have become quite complex. A re-design of the databases would immediately force a rewrite of all application logic and would be, of course, an economic impracticality. Re-engineering for client-server on those terms was out of the question for us. Nevertheless, our user community, like users everywhere, had discovered the world of personal computing and had begun to clamor for freer access to their data, more user-friendliness in the applications and interaction with the numerous off-the-shelf decision support tools available to them.

We feel that this tide of enthusiasm and end-user involvement will not be turned back and that any sign of reluctance to participate on our part, however well-intentioned or soundly based in systems theory, will be perceived as a negative. Complexity is at the very heart of the data processing industry and we believe that it is our responsibility to respond to our users' needs and desires with imaginative yet sound business computing solutions, regardless of the degree of difficulty involved. It was with that attitude and an almost total lack of understanding of even the most basic client-server terminology that we attended the first HP technology camp over 5 years ago, in January of 1991, held in Cupertino, California.

Setting the Corporate Objectives

From what we learned in those early seminars and meetings we were able to develop our corporate client-server migration objectives. Simply stated, we needed to **find a safe, secure and supportable way to migrate our large and complex COBOL Image and View-based legacy applications to a PC-based client-server model on a step-by-step basis, simultaneously preserving the old while gradually introducing the new, one module and one program at a time, utilizing, to the fullest extent possible, only industry compatible tools and techniques.**

Since that time I have been involved in client-server research and product evaluations. My assignment was to design the development infrastructure and standards within which we would be able to accomplish these ambitious objectives without losing control of our business or the confidence placed in us by our users.

By honoring the mandate, we are able to see many immediate benefits, including:

- Eliminating the need for an up-front re-design of the system. Even if this was a practical thing to consider, it would be best not to attempt at a time when, as a company, we are at the lowest point on the client-server learning curve. The knowledge gained during the migration will surely pay dividends in the future when the inevitable re-engineering effort actually does take place.

- Protecting our users' sizable investment in HP hardware by being able to coexist with the terminal-based software while gradually introducing the PC-based system, one seat at a time.

- Greatly lessening the impact of change and training on our end users and on our customer service and product support personnel. Because the new and the old can coexist, the pressure is off the learning cycle, allowing the early pace to be set by the individual learner. This is very important in a nationwide environment such as ours.

- Protecting our investment in man hours of program development by using, to the fullest extent possible, only industry compatible tools and techniques. We are not being naïve on this topic and we realize that this is an imperfect and developing environment, but we also know that time is of the essence from a user perception point of view and that by sticking to our self-imposed standards as closely as possible, we are a lot closer to the goal of 'openness' than if we allow our programming effort to take off in multiple directions at the outset. Through the careful use of 'middleware' products we have been able to modularize the host and database access functions so that our programs contain, with minor syntactical variations, only standard coding while achieving 'industrial strength' performance.

Selecting the Relational Database Engine

It is obvious that the ability to coexist with the existing software and hardware is vital to the feasibility of our client-server migration project. It also means, however, that our underlying TurboImage database structure cannot be replaced for the duration of the migration effort (a process we estimate will take between 3 and 5 years to fully complete). Our obvious (and only) choice for a relational database engine was the Image/SQL product from Hewlett Packard.

This strategically important product enables us to gradually transform our programs into an industry standard SQL-based relational format without disrupting the TurboImage core upon which all the rest of the system depends. Only after this transformation is complete will we be able to consider using another database product.

Selecting the PC Operating System and Network Protocol

Because this is our first official foray into the world of PC-based applications and at the time of its introduction we had not yet made our first production installation, we felt it would be best to start at the most current OS level, so we have selected Windows 95. TCP/IP support is bundled with Windows 95 so it was the obvious choice for our network protocol stack, greatly simplifying the Host/PC network connection to MPEix.

Selecting the Software Development Language

We evaluated a number of promising software development tools and languages with the hope of finding one that could handle all of our programming requirements. Again guided by our management mandate to use, wherever possible, only industry-compatible tools and methods, we took a special interest in Microsoft Visual Basic (Professional Edition). Out of the box, this product appeared to offer the syntactical strength, support for the graphical user interface and database access capabilities that we felt were necessary for a successful conversion of our software. We were somewhat concerned by rumors we had heard about the execution speed of Visual Basic programs, however, so we reserved the option of developing some of the more code-intensive routines in C++. As it turned out, we never found it necessary to invoke our C++ safety net because, in the world of graphical user interfaces, event-driven programming techniques and host database access routines, the Visual Basic execution speed on a modern PC is a good match. Certainly in terms of long term supportability, simplicity and standardization of code, we feel that any trade-off in terms of faster program execution speed or, within limits, PC resource utilization, is a compromise well worth making. All of our application programs, written exclusively in Visual Basic, execute crisply and responsively. In a user-driven environment, any other run-time considerations become academic. I would not want to say that we will never develop any lower level code modules but at this time it appears to be highly unlikely.

Making the PC a Team Player

Regardless of the development tool set in question, throughout our evaluation process we found a consistent and serious lacking in the area of support for multiple-programmer development environments. This included such things as code library functions, source file management, version control, audit capabilities against accidental or intentional file corruption, group compilations, etc. This was no less true with Visual Basic than with the others.

It is imperative to structure and define the development, testing, debugging and production environments before any serious software development takes place. Postponement of this important step will likely result in an out of control situation early and often in the project life.

There is probably enough material here to warrant a separate paper and there are certainly plenty of software products out there attempting to deal with these issues, including the bundled Source Safe product from Microsoft, but suffice it to say that we found it more advantageous to develop our own server-based Source File Librarian, loosely patterned after the discontinued Microsoft Delta product, than to try to adapt to the complex rule bases imposed by the off-the-shelf solutions we tried. This program, like all of our programs to date, was developed strictly in Visual Basic. It is simple to use, is tailored to its specific task and has enabled us to move forward in a controlled fashion with a startup force of four full-time programmers and two system designers.

Project Definition -- Batch Job Front Ends

Rather than to initially attempt a full-blown SQL-based rewrite of our off-line reports, we have decided to front-end each of the jobs with a small Visual Basic program launched by, and under control of, the Application Manager. By choosing this path we are able to put a new face on our software in a relatively short time frame while simultaneously developing a repository of GUI-based programming skills and experience. Because the existing COBOL programs are ultimately executed under this scheme, we are able to postpone the development of the more intense database navigation and update routines, thus allowing some additional time for the supporting software tools, such as Image/SQL, to stabilize and mature.

We have found that it is very easy to underestimate the steepness of the learning curve for both new and experienced programmer/analysts and we have learned to take the conservative view wherever relational DBMS features or performance issues are involved. The ultimate redesign of the off-line reports for full SQL compliance will be in the second phase of our project.

Project Definition -- Interactive Modules

The inquiry, posting and file maintenance portions of our system, because of their complete dependency on the Block Mode terminal features, do not correlate well to the Windows environment. Our early attempts at transposing these modules directly to the GUI environment have yielded mixed results at best. Event driven programming requirements and limited screen 'real estate', when compared to the dumb terminal interface, define the need to completely redesign these applications for Windows compatibility at the outset.

Again trying to allow as much time as possible for the more intense Image/SQL features such as row locking and transaction control to develop and mature, we have decided to begin our interactive rewrite with the Inquiry modules. Building upon the foundation of Visual Basic programming techniques developed during the batch job front-end phase, our programmers will now be able to focus on the issues of SQL database access and recordset manipulation. It is our hope that, by this time, the techniques associated with screen handling and event coding will have become more of a background concern for both our programmers and our system design team. Here, again, a respect for the steep learning curve can go a long way toward minimizing frustration and improving programmer productivity in the long run.

Design and implementation of the posting and file maintenance routines will follow the successful deployment of the inquiry modules with the objective of again building upon the knowledge gained and skills developed during the inquiry rewrite.

Project Definition -- The Application Manager

In conjunction with all of the system elements described above and the enabling technologies (middleware) described below, we developed the Application Manager PC-resident control program. Written entirely in Visual Basic, Application Manager is the centerpiece of our client-server system. It connects our ODBC compliant Visual Basic programs, executing from either a LAN-connected Windows workgroup or individual LAN-connected workstation, with MPEix and the existing TurboImage databases on the host. Properly authorized users are able to move freely between batch and interactive functions on any of the three SEDC software sub-systems. They may switch between the old and the new code versions, as required, while security remains the responsibility of the host system. By using only standard Windows controls and methods, we have been able to give all of our programs, including Application Manager, a consistent 'look and feel'.

The high degree of modularity built in to our system design permits us to release our product on a program-by-program basis. Remembering that the end-user learning curve is also a steep one, these system characteristics were designed to make it easier for them to acclimate themselves to the new methodology confidently and, at least initially, at their own pace.

The Missing Link

Having selected the host database engine, the network protocol, the PC operating system, the programming language and the source file librarian, we needed only to decide upon a middleware piece to bring the disparate Host and PC worlds together.

It soon became obvious that to truly present a complete PC-based client-server solution to our users, we were going to need a lot more than the ability to read and write the host database from the Visual Basic programs. We would need, in fact, some sort of programmable message-level interface to the 3000 for logon security, batch job front-end setups, job control and spool file management. To keep within the management-mandated safety and security guidelines, we wanted all security issues to continue to be handled by MPEix on the 3000, realizing that any information maintained on the PC must realistically be considered to be public information. For this message-level interface functionality we selected the LeeTech Software, Inc. Client-Server Foundation (CSF) middleware product. CSF allows us to seamlessly connect, via four simple DLL function calls, to our existing User Menu Security system by attaching the workstation's Application Manager program to our COBOL-written message handler residing on the host. The message formats and functions were designed by us to accommodate security, job and spool file issues. Our green bar Print Preview feature for off-line reports has been very well received by our end-users and has provided both an added value and an incentive for them to implement and learn the new software quickly. In addition, browsing the report and standard list files has turned out to be an effective low-stress learning method for Windows/Mouse beginners.

We also discovered that while standard ODBC with SQL pass-thru to Image/SQL gave our Visual Basic programs logical access to our existing TurboImage databases, it seriously lacked in robustness, performance and flexibility to the point where, after much evaluation, we felt that it was not suitable for use in our project at this time. To solve this problem, we again selected a LeeTech Software, Inc. middleware product called SQL Developer Kit or SDK. This product provides direct access to the Host-resident TurboImage databases without invoking either the Microsoft Jet Engine routines or the ODBC drivers. Used exclusively in our interactive module rewrite and soon to be used in phase 2 of our batch job redesign, SDK has proven to be a high-speed, reliable, industrial-strength SQL database interface, portable across a variety of popular industry platforms and relational database products.

Avoiding Third Party Dependency

A major concern whenever a third party product is introduced is that we might become so dependent upon the intrinsic functionality of that product that we wind up creating just another hard-coded, proprietary, ‘closed’ system. We, too, were concerned about this phenomenon as it pertained to our middleware selection. Instead of backing away from the tool, however, we used the tool itself to develop a set of library routines which allow us to write our programs as if they were standard ODBC applications, replacing the Visual Basic record set statements on a one-for-one basis with almost identical function and sub-function calls. Should it ever become necessary, desirable or practical for us to switch back to a pure ODBC solution, we are basically only a compile away, after some very minor syntactical changes to convert our record set functions back to standard Visual Basic statements.

This is the least obvious feature of our system but is also one of its major strengths, for it allows us to confidently go forward with our development, using a truly robust database interface, without having to compromise our stated objectives concerning future operating system and database independence, industry standardization and upward compatibility of code.

Regardless of the timing or direction of future ODBC development, the LeeTech/SDK will continue to shield us from the intricacies of the various vendors’ implementations by allowing us to simply insert a different SDK ‘engine’ between our standard Visual Basic code and the Host operating system or database environment. This is an oversimplification to some extent but, as long as we avoid the use of vendor-specific SQL extensions in our code, the concept is sound.

It would be foolish to attempt to predict the future of the industry today but we feel that with all of our Visual Basic code remaining ODBC compliant and the LeeTech/SDK serving as our database interface, we are well positioned to respond to future changes in the ODBC/Relational Database paradigm.

Conclusion

Much work remains to be done and we in no way think that ours is the only viable answer or that we have examined every possibility in this new world of possibilities. I offer this paper simply as a means of telling our story and in the hope that it may contain something of value to you. The enthusiastic acceptance exhibited by our user community gives rise to the feeling that, at least within our own purview, we are on track for an overwhelming success.

We hope that our presentation today will effectively demonstrate that with thorough planning, careful selection of tools and strategic use of solid middleware products, all of the pieces can be brought together, enabling all of us to meet our client-server migration objectives in an orderly, cost-effective and supportable manner.